A Fast Counterexample Minimization Approach with Refutation Analysis and Incremental SAT

Shengyu Shen
School of Computer Science, National University of Defense Technology
ChangSha, China 410073
Tel : +86 0731 4576143
Fax : +86 0731 4511529

Ying Qin
School of Computer Science, National University of Defense Technology
ChangSha, China 410073
Tel : +86 0731 4573681
Fax : +86 0731 4511529

SiKun Li
School of Computer Science, National University of Defense Technology
ChangSha, China 410073
Tel : +86 0731 4575981
Fax : +86 0731 4511529

Abstract— It is a hotly research topic to eliminate irrelevant variables from counterexample, to make it easier to be understood. BFL algorithm is the most effective Counterexample minimization algorithm compared to all other approaches, but its run time overhead is very large due to one call to SAT solver per candidate variable to be eliminated. So we propose a faster counterexample minimization algorithm based on refutation analysis and incremental SAT. First, for every UNSAT instance of BFL, we perform refutation analysis to extract the set of variables that lead to UNSAT, all variables not belong to this set can be eliminated simultaneously. In this way, we can eliminate many variables with only one call to SAT solver. At the same time, we employ incremental SAT approach to share learned clauses between similar instances of BFL, to prevent overlapped state space from being searched repeatedly. Theoretic analysis and experiment result shows that, our approach can be 1 to 2 orders of magnitude faster than BFL, and still retain the minimization ability of BFL.[1]

I. INTRODUCTION

Model checking technology is widely employed to verify software and hardware system. One of its major advantages in comparison to such method as theorem proving is the production of a counterexample, which explains how the system violates some assertion.

However, it is a tedious task to understand the complex counterexamples generated by model checker. Therefore, how to automatically extract useful information to aid the understanding of counterexample, is an area of active research[1][2].

If we can extract a subset of variables that are sufficient to lead to counterexample, then these variables can express a large number of counterexamples, not just the individual one generated by model checker. In the remainder of this paper, we call this variable subset as **minimization set**, and call the process that extract minimization set as **counterexample minimization**.

[1]

K Ravi[1] proposes a counterexample minimization algorithm called Brute Force Lifting algorithm (BFL). For every free variable v, BFL constructs a SAT instance SAT(v), to determine if v can prevent the counterexample. If SAT(v) is UNSAT, then v is irrelevant to counterexample, and can be eliminated. K Ravi compares BFL with other counterexample minimization approaches, and concludes that BFL is the most efficient one, it can often eliminate up to 70% free variables. However, the time complexity of BFL is much more higher than other existing approaches due to the following reasons:

1 It need one call to SAT solver per candidate variable to be eliminated;
2 It can't share learned clause between similar SAT instances, so overlapped state space may be searched repeatedly.

Accordingly, the key to reduce time overhead of BFL are:

1 Eliminate multiple variables after every call to SAT solver;
2 Share learned clauses between similar SAT instances, to avoid searching overlapped state space repeatedly.

Therefore, we propose a faster counterexample minimization approach in this paper, which employ refutation analysis for all UNSAT instances to extract all relevant variables and eliminate all irrelevant variables. At the same time, our approach employs incremental SAT to share learned clauses between similar instances.

We implement our algorithm based on zchaff [5] and NuSMV[4], and perform experiment on ISCAS89 benchmark suite[6]. Experiment result shows that, our approach can be 1 to 2 orders of magnitude faster than BFL algorithm and with minor lost in its minimization ability.

The remainder of this paper is organized as follows. Section II presents background material. Section III presents the refutation analysis algorithm. Section IV presents the incremental SAT approach. Section V present experiment result of our approach and compare it to that of BFL [1]. Section VI reviews related works. Section VII concludes with a note on future work.

II. PRELIMINARIES

A. *Bounded Model Checking*

We first define the Kripke structure:

Definition 1 Kripke structure is a tuple $M=(S, I, W, T, A, L)$, with a finite set of states S, the set of initial states $I \subseteq S$, the input variable set W, transition relation between states

T:$S\times W\times S\rightarrow\{0,1\}$, and the labeling of the states L:$S\rightarrow 2^A$ with atomic propositions set A.

Bounded Model Checking (BMC)[3] is a model checking technology that consider only limited length path. We call this length as the bound of path. We denote the state of the i-th and (i+1)-th cycle as S_i and S_{i+1}, and transition relation between them as $T_i(S_i,W_i,S_{i+1})$, with input variable set of i-th cycle denoted by W_i.

Let the safety assertion under verification be *ASSERT*, the goal of BMC is to find a state S that violate *ASSERT*, that is to say, $\neg ASSERT\in L(S)$. Lets denote $\neg ASSERT$ as P.

Let P at i-th cycle as P_i ,then BMC problem can be expressed as:

$$F = I \wedge \bigwedge_{0\le i<k} \neg P_i \wedge P_k \wedge \bigwedge_{0\le i<k} T_i(S_i,W_i,S_{i+1}) \qquad (1)$$

Reduce equation (1) into SAT instance, and solve it with SAT solver, then a counterexample can be found if it exists.

B. BFL Algorithm and its Shortcoming

BFL algorithm proposed by K Ravi[1] can eliminate much more free variables than all existing algorithm, often up to 70% free variables can be eliminated.

We give some terminology below:

Definition 2: Assume the bound of counterexample is k, and denote the set of free variables as $Free = I \cup \bigcup_{0\le i<k} W_i$. The assignment to variable v in counterexample is denoted by Assign(v), the assignment to variable set V in counterexample is denoted by Assign(V)={Assign(v)|$v\in V$}.

Obviously, the set *Free* includes input variables at all cycles and initial state variables.

For a free variable $v\in Free$, v is an irrelevant variable if and only if the following statement hold true: "no matter what value do v take on, it can't prevent the counterexample from happen. That is to say, it can't prevent P_k of equation (1) from equal to 1". Formal definition of irrelevant variable is given below:

Definition 3 Irrelevant Variable: for $v\in Free$, v is an irrelevant variable iff:

$$\neg\,\exists c\in\{0,1\}.[[M]]_k \wedge (v\Leftarrow c) \wedge A \wedge \neg P_k \qquad (2)$$

where A=($Free-\{v\}\Leftarrow$ Assign($Free-\{v\}$))
and $[[M]]_k=\wedge_{0\le i<k}T_i(S_i,W_i,S_{i+1})$

Convert equation (2) into SAT instance, then v is irrelevant variable iff this SAT instance is Unsatisfiable.

Thus, the BFL algorithm that extracts minimization set from counterexample is show below:

Algorithm 1:BFL Algorithm

1 $F''=[[M]]_k\wedge\neg P_k$
2 foreach $v\in Free$
  a) $F'=F''\wedge(Free-\{v\}\Leftarrow$Assign($Free-\{v\}$))
  b) if(SAT_Solve(F')==UNSAT)
    i. $Free=Free-\{v\}$
3 *Free* is the minimization set

To make it more distinct, we give the following 2 definitions:

Definition 4 Model Clause Set: all clauses generated from F'' in step 2a) of algorithm 1.

Definition 5 Assignment Clause Set: all clauses generated from **(** $Free-\{v\}\Leftarrow$ Assign($Free-\{v\}$) **)** in step 2a) of algorithm 1.

We call the former as model clause set, because F'' represent inverted model checking problem of equation (1). We call the latter as assignment clause set because they are used for assigning to all variables their value in counterexample, except v. For every $v'\in Free-\{v\}$, its assignment clause contain only one literal. If Assign(v')==1, then assignment clause of v' is $\{v'\}$, otherwise it is $\{\neg v'\}$. SAT solver will assign these values to them by BCP when solving this instance.

III. COUNTEREXAMPLE MINIMIZATION WITH REFUTATION ANALYSIS

In this section, we first describe the overall algorithm flow in subsection A, and then describe the most important part – refutation analysis in subsection B. We will prove its correctness in subsection C.

A. *Overall algorithm flow*

BFL algorithm can eliminate much more variables than all existing algorithm, but its time overhead is too high. Therefore, it is very important to reduce its timing overhead.

Overall flow of our algorithm is show by algorithm 2:

Algorithm 2 BFL algorithm with refutation analysis

1 $F''=[[M]]_k\wedge\neg P_k$
2 foreach $v\in Free$
  a) $F'=F''\wedge(Free-\{v\}\Leftarrow$Assign($Free-\{v\}$))
  b) if(SAT_Solve(F')==UNSAT)
    i. R''=Refutation_Analysis()
    ii. $Free=Free\cap R''$
3 *Free* is the minimization set

Compare it to algorithm 1, step 2b) of algorithm 2 are newly inserted steps, which is highlighted with bold font. In step 2b)i, we perform refutation analysis to extract the variables set R" that lead to UNSAT. And then in step 2b)ii, we eliminate all variable not belong to R".

B. *Refutation Analysis*

As stated by last subsection, we perform refutation analysis to extract the variable set R'' that lead to UNSAT.

For a SAT instance F', we denote its model clause set by F'', and its assignment clause set by A. After SAT solver finished running, denote its learned clause set as C.

If result of F' is UNSAT, then there must be a conflict clause at decision level 0, we denote it by c. Because the decision level is 0, so there are no decided variables, any variables can only take on their value by implication.

Staring from clause c, we can traverse the implicate graph in reverse direction, to obtain the set of origin clauses S that lead to conflict.

We denote the assignment clauses in S by $S\cap A$, then the set of variable that lead to UNSAT is R''={v| $\{v\}\in S\cap A\}\cup\{v|\ \{\neg v\}\in S\cap A\}$.

Now we present the refutation analysis algorithm below, the correctness proof will be given in next subsection.

Algorithm 3 Refutation Analysis

1 set $S=\phi$;
2 queue $Q=\phi$;

3 foreach literal $l \in c$
   a) push antecedent clause of l into Q
4 while(Q is not empty)
   b) cls=pop first clause from Q
   c) if(cls is a unit clause)
      i. $S=S+\{cls\}$
      ii. If(cls is a learned unit clause) return R=Free-{v}
   d) else
      i. foreach literal $l \in cls$
         1. assume $ante$ is antecedent clause of l
         2. if($ante$ has not being visited)
            a) push $ante$ into Q
            b) mark $ante$ as visited
5 $R''=\{v|\{v\} \in S \cap A\} \cup \{v|\{\neg v\} \in S \cap A\}$ are variables lead to UNSAT

There is a special case in step 4b)ii, when cls is a learned unit clause, we can't backtrack further because the SAT solver has not record the clauses involved in resolution to construct cls. In this case, we abandon the effort to extract R'', and simply return $R''=Free-\{v\}$. This means that we can eliminate only one variable v in this case.

Fortunately, we have not met with this special case in experiments. I suspect that this is because of the 1UIP conflict learning mechanism, which never generate learned unit clause.

C. *Correctness Proof*

We prove the correctness of algorithm 2 and 3 with following theorem:

Theorem 1: $F'' \wedge \wedge_{cls \in S} cls$ is an Unsatisfiable clause subset of F'

Proof: it is obvious that $F'' \wedge \wedge_{cls \in S} cls$ is a clause subset of F', so we only need to prove that it is Unsatisfiable.

Assume C'⊆C is the set of learned clauses met with by algorithm 3 while traversing implication graph. Thus, $F'' \wedge \wedge_{cls \in S} cls \wedge \wedge_{cls \in C'} cls$ is Unsatisfiable. Then if we can remove $\wedge_{cls \in C'} cls$ from it, and still retain its unsatisfiability?

For every learned clause $cls \in C'$, assume $NU(cls)$ and $U(cls)$ are non-unit clauses set and unit clauses set that involved in resolution to construct cls.

It is obvious that $NU(cls) \in F''$. And according to [8], unit clauses never involve in resolution, so $U(cls)$ is empty set. So we can remove $\wedge_{cls \in C'} cls$ from $F'' \wedge \wedge_{cls \in S} cls \wedge \wedge_{cls \in C'} cls$, and still retain its unsatisfiability

Thus this theorem is proven. □

Theorem 2: $F'' \wedge \wedge_{v' \in R''} (v \Leftarrow Assign(v'))$ is an Unsatisfiable clause subset of F'

Proof: it is obvious that $F'' \wedge \wedge_{v' \in R''} (v \Leftarrow Assign(v'))$ is an clause subset of F'. Thus we only need to prove that $F'' \wedge \wedge_{v' \in R''} (v \Leftarrow Assign(v'))$ is Unsatisfiable.

According to algorithm 3, $\wedge_{v' \in R''} (v \Leftarrow Assign(v'))$ is equal to $\wedge_{cls \in S \cap A} cls$. So we only need to prove that $F'' \wedge \wedge_{cls \in S \cap A} cls$ is Unsatisfiable.

According to theorem 1, $F'' \wedge \wedge_{cls \in S} cls$ is Unsatisfiable, which can be rewritten as $F'' \wedge \wedge_{cls \in S \cap A} cls \wedge \wedge_{cls \in S-A-F''} cls$.

Lets discuss it in two aspects:

1 If $S-A-F''$ is empty set, then $F'' \wedge \wedge_{cls \in S \cap A} cls$ is Unsatisfiable
2 Otherwise, $S-A-F''$ isn't empty set. In this case, algorithm 3 will meet with a learned unit clause. According to step 4b)ii of algorithm 3, it will abandon the effort to extract R'', and eliminate only one variable v. In the case, $F'' \wedge \wedge_{cls \in S \cap A} cls$ is Unsatisfiable

Thus this theorem is proven. □

D. *Complexity Analysis*

We know that the space overhead of refutation analysis mainly reside in set S and queue Q. Lets analyze as below:

- We add a tag to each clause in clause database of SAT solver, to indicate that if this clause belongs to set S. Therefore, space overhead of S is linear to size of clause database.
- For queue Q, it may contain learned clauses. Because conflict analysis algorithm of SAT solver also need to perform similar implicate graph traversing, so space overhead due to Q is not larger than that of SAT solver.

Next, let's analyze the time complexity of our algorithm.

In algorithm 3, the most complex part is the if statement in step 4c)i2. For every clause that has been in Q, this if statement will be run once. Because the size of Q is much smaller than clause database, so time overhead of algorithm 3 is much smaller than that of SAT solver.

In algorithm 2, one call to refutation analysis algorithm will eliminate many irrelevant variables, thus prevent them from calling SAT solver.

We will present the number of call to refutation analysis and SAT solver in experiment result.

IV. Incremental SAT

From step 2a) of algorithm 2, we can deduce that model clause set generated from F'' is independent of the loop of step 2. So we can add model clause set into clause database before that loop. In this way, we do not need to add model clause set into clause database repeatedly.

At the same time, we also do not need to add assignment clause set into clause database in each iteration of the loop in step 2 of algorithm 2. Assume that in two back-to-back iterations, we need to process $v1$ and $v2$ in these two iterations, so we only need to insert assignment clause of $v1$ in clause database after the first iteration, and delete assignment clause of $v2$ before the second iteration.

Therefore, we modify algorithm 2 into the following algorithm 4. All new steps are highlight with bold font.

Algorithm 4 BFL algorithm with refutation analysis and Incremental SAT

1 $F'' = [[M]]_k \wedge \neg P_k$
2 Insert $F'' \wedge (Free \Leftarrow Assign(Free))$ into clause database
3 Foreach $v \in Free$
   a) Delete assignment clause of v from clause database
   b) if(SAT_Solve(F')==UNSAT)
      i. R''=Refutation_Analysis()
      ii. $Free = Free \cap R''$
      iii. For all $v' \in Free-R''$, delete its assignment clause from clause database
   c) Else
      i. Insert assignment clause of v into clause database
4 $Free$ is the minimization set

Here the readers may doubt that: in step 3a) and 3b)iii, when deleting assignment clauses, if we need to delete relevant learned clauses?

N. Een[7] conclude that: for a certain SAT instance, when deleting a clause that contain only one literal, all learned clause can be safely kept in clause database.

This statement improve performance in 2 aspects:

1 Learned clause can be share between similar instances

2 No need to waste time on deleting learned clauses.

V. Experiment Result

K Ravi [1] only presents the circuits that used to generate counterexample, but has not presented the assertion used. Therefore, we cannot compare our approach with his one directly. So we implement his algorithm and ours in zchaff[5], such that we can compare them with same circuits and assertions.

We perform counterexample minimization with BFL [1] and ours. The timeout limit is set to 20000 seconds.

Experiment result is presented in table 1. The first column is the circuit used to generate counterexample. The second column presents the length of counterexample. The third column presents number of free variables.

The 4th column is the number of irrelevant free variables eliminated by K Ravi's BFL[1]. Divide this number with the third column, we can then get the minimization rate in the 5-th column, run time of BFL is shown in 6-th column.

The 7-th column is the number of irrelevant free variables eliminated by our approach. Divide this number with the third column, we can then get the minimization rate in the 8-th column, run time of ours is shown in 9-th column. The speedup compared to BFL is shown in last column.

In table 2, we present run time statistics of our algorithm:

The 1st column is the name of circuits. Their number of variables and clause are presented in 2nd and 3rd column. Their number of free variable are presented in 4th column, the variable eliminated by refutation analysis in step 3b)ii of algorithm 4 are presented in column 5. The numbers of UNSAT instance are presented in the 6th column. The numbers of SAT instance are presented in 7th column. The peak size of Q and S are presented in last 2 columns.

TABLE I
Experiment Result

Circuits	CE length	Free Vars	Result of BFL[1]		Result of our approach		
			Eliminated Vars	Run time	Eliminated Vars	Run time	Speedup
s1512	21	667	606	39.85	606	3.45	11.55
s1423	24	483	400	292.65	397	7.12	41.10
s3271	15	507	432	70.16	432	6.11	11.48
s3384	13	743	615	126.1	615	9.61	13.12
s3330	6	373	295	19.69	295	1.77	11.12
s5378	10	530	416	133.27	411	8.21	16.23
s9234	7	362	226	75.37	226	10.36	7.28
s13207	22	1352	1109	>20000	1093	153.92	>100
s38584	14	1621	1069	>20000	1069	682.19	>10
s38417	14	2029	980	>20000	981	947.84	>10

By the way, BFL of s13207,s38584 and s38417 don't finish within 20000 seconds, its variables is eliminated by incremental SAT approach without refutation analysis

TABLE 2
Run Time Statistics of Our Algorithm

Circuits	Vars	Clauses	Free Vars	Eliminated by ref analysis	Num of UNSAT	Number of SAT	Peak size of S	Peak size of Q
s1512	14858	39735	667	601	5	61	3219	397
s1423	16565	44248	483	369	29	85	8250	736
s3271	21769	59656	507	416	16	75	5042	448
s3384	19452	50353	743	596	19	128	7127	458
s3330	6935	17322	373	278	17	78	2308	321
s5378	16180	42415	530	387	24	119	4253	484
s9234	18291	49555	362	173	53	136	6454	581
s13207	107079	284839	1352	1025	68	259	34108	1999
s38584	237756	661828	1621	1005	64	552	73971	5119
s38417	211653	576324	2029	910	71	1048	81855	7703

From this two table, we can conclude that:

1 Our approach is 1~2 orders of magnitude faster than BFL;

2 Our approach doesn't lose the minimization ability.

3 From 5th column of table 2, we can see that most variable are eliminated by refutation analysis, and do not need to run SAT solver for them any more

4 Compare last 2 columns of table 2 to 3rd column, the size of Q and S are much smaller than that of clause database

VI. Acknowledge

This research work is Supported by the National Natural Science Foundation of China under Grant No. 90207019;the National High Technology Development 863 Program of China under Grant No. 2002AA1Z1480.

VII. Conclusion

To make the counterexample easier to be understood, irrelevant variables must be eliminated. BFL is the most effective counterexample minimization algorithm. However, its time overhead is too large.

Therefore, we propose a faster counterexample minimization algorithm in this paper. Our algorithm run 1 to 2 magnitude orders faster than BFL, and with only minor lost in minimization ability.

In this paper we only deal with safety assertion, we would also like to address minimization approach for loop like counterexample of liveness property in future work.

References

[1] K Ravi and Fabio Somenzi. Minimal Assignments for Bounded Model Checking. In TACAS'04,pages 31-45,2004. LNCS 2988.

[2] H.Jin, K.Ravi,and F.Somenzi. "Fate and free will in error traces".TACAS'02,pp 445-458,2002.LNCS 2280.

[3] A. Biere, A. Cimatti, E.M. Clarke, M. Fujita, Y. Zhu . "Symbolic Model Checking using SAT procedures instead of BDDs".In DAC'99.pages 317-320, 1999.

[4] A. Cimatti, et.al "NuSMV 2: An OpenSource Tool for Symbolic Model Checking". In CAV'02,pages 359-364, 2002.LNCS 2404

[5] M. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efficient SAT solver. In DAC'01, pages 530-535,2001.

[6] http://www.cbl.ncsu.edu/CBL_Docs/iscas89.html

[7] N. Een and N. Sorensson. Temporal Induction by Incremental SAT Solving. In BMC'03.

[8] L.Zhang, C.Madigan, M.Moskewicz, and S.Malik. Efficient conflict driven learning in a Boolean satisfiability solver. ICCAD 2001.